IN THE CLAIMS:

Claims 1 – 30 (Canceled).

Claim 31 (Currently Amended): A method, in a mixed static and dynamic environment, for a virtual machine in which statically precompiled code may be securely executed by a virtual machine by means of a compiler or code generator, the method comprising the steps of:

a) saving pre-compiled programs, including determining where to place said programs, annotating the programs with dependent information, annotating the programs with dependence information, and processing the programs to produce a further annotated executable code with annotations to help adapt the code to a new executable environment;

b) verifying that an intermediate ~~or byte-code~~code representation of the program is safe;

c) forming a secure hash describing the ~~generated~~precompiled code;

d) forming a secure hash describing the ~~byte~~intermediate code representation;

e) digitally signing the secure hashes of the ~~generated~~precompiled code and the ~~byte~~ intermediate code representation; the executing virtual machine reuses ~~this code~~the precompiled code and the intermediate code representation by

f) verifying that the secure hash of the ~~byte-code~~intermediate code representation matches the digitally signed secure hash for the ~~byte-code~~intermediate code representation;

g) verifying that the secure hash of the ~~generated~~precompiled code matches the digitally signed secure hash for the ~~generated~~precompiled code; and

h) loading and executing the ~~generated~~precompiled code[[.]];

wherein the step of annotating the programs with dependence information includes the steps of annotating the programs with fine-grain dependencies, and processing said fine-grain

dependencies by a dependence granularity adjuster to replace some fine-grain dependencies by coarser-grain dependencies to produce a final list of dependence annotations.

Claim 32 (Currently Amended): A method, in a mixed static and dynamic environment, for linking separately statically, precompiled code at run-time within a virtual machine by modifying the code, the method comprising the steps of

using a compiler to perform the steps of

a) saving pre-compiled programs, including determining where to place said programs, annotating the programs with dependent information, annotating the programs with dependence information, and processing the programs to produce a further annotated executable code with annotations to help adapt the code to a new executable environment;

b) maintaining symbolic entries for externally referenced symbols; and

c) maintaining a mapping from locations in the ~~generated~~precompiled code that reference external symbols to the symbolic entry for that symbol;

and the virtual machine, before the code is executed, performs the steps of

d) using the mapping and symbolic entries created by the compiler to generate direct references in the ~~generated~~precompiled code to the externally referenced symbols that have been resolved by the virtual machine; and

e) performing ~~the~~a given default action on those external symbols that have not been resolved.

wherein the step of annotating the programs with dependence information includes the steps of annotating the programs with fine-grain dependencies, and processing said fine-grain dependencies by a dependence granularity adjuster to replace some fine-grain

dependencies by coarser-grain dependencies to produce a final list of dependence annotations.

Claim 33 (Currently Amended): A method, in a mixed static and dynamic environment, for updating statically generated precompiled code (C), at run-time, when separately compiled code (S), which contained symbols referenced by C changes, the method comprising the steps of:

saving pre-compiled programs, including determining where to place said programs, annotating the programs with dependent information, annotating the programs with dependence information, and processing the programs to produce a further annotated executable code with annotations to help adapt the code to a new executable environment;

having a compiler generating the code for S to a) associate with method and data names, or signatures, in S a secure hash of the name of the method and data names or signatures in S with the compiler for C recording the secure hash for the byte code corresponding to any S that affects the code generated for C; and

having the virtual machine executing C to

b) check if any byte codes associated with any names in the S relied upon by C have changed by comparing the secure hash of the names associated with S with the secure hash stored for this byte code in C, and

c) dynamically recompile the byte codes associated with C if any byte codes associated with S have changed[[.]];

wherein the step of annotating the programs with dependence information includes the steps of annotating the programs with fine-grain dependencies, and processing said fine-grain dependencies by a dependence granularity adjuster to replace some fine-grain

dependencies by coarser-grain dependencies to produce a final list of dependence annotations.

Claim 34 (Currently Amended): A method, in a mixed static and dynamic environment, for maintaining full compliance with a language requiring dynamic compilation without requiring the overhead of a just-in-time compiler to be present in the virtual machine, while enabling the use of statically generated precompiled code (C) for some byte code that depends on some byte code (S) that may be separately compiled,

saving pre-compiled programs, including determining where to place said programs, annotating the programs with dependent information, annotating the programs with dependence information, and processing the programs to produce a further annotated executable code with annotations to help adapt the code to a new executable environment;

having a compiler generating code for S

a) associate with S a secure hash of the byte code associated with S;

having the compiler for C to

b) record the secure hash for the byte code corresponding to any S that affects the code generated for C; and

having the virtual machine executing C to

c) check if any byte codes associated with any code S relied upon by C have changed by comparing the secure hash of the byte code associated with S with the secure hash stored for this byte code with C; and

d) interpret the byte codes corresponding to C[[.]];

wherein the step of annotating the programs with dependence information includes the steps of annotating the programs with fine-grain dependencies, and processing said fine-grain dependencies by a dependence granularity adjuster to replace some fine-grain dependencies by coarser-grain dependencies to produce a final list of dependence annotations.

Claim 35 (Previously Presented): A method according to claim 31, comprising the further step of the compiler performing dependence checks during program execution to avoid using a stale code for a procedure in the event of changes to other codes.

Claim 36 (Previously Presented): A method according to claim 35, wherein the step of performing dependence checks includes using time stamps to determine if any of the classes on which the code is dependent have changed.

Claim 37 (New): A method according to claim 31, comprising the further step of using a QSI recorder to process a given class of compiled procedures, including the steps of

a) examining each of the procedures in the class,

b) for any of said procedures not compiled with a given optimization level, compiling said procedures with said given optimization level,

c) creating a QSI for the class, said QSI including a header region,

d) storing information is said header region, said information including a predetermined constant identifying the QSI as a QSI, information identifying a version of the virtual machine, information identifying an operating system version, and a target architecture,

e) recording a timestamp identifying the time of creation of the QSI,

f) recording additional information to identify a loaded class, said additional information including a fully qualified name of the class, and a defining class loader of the class,

g) determining whether said defining class loader is a primordial class loader,

h) if the defining class loader is not a primordial class loader, then storing said defining class loader as a digest of a class file, thereby to enable the virtual machine to check, during a program execution, whether a class was defined by a given class loader during offline compilation and execution,

i) recording a list of other classes on which code in the QSI is dependent,

j) creating a directory containing pointers to a plurality of procedure codes,

k) writing said procedure codes and related auxiliary information to the QSI, said related auxiliary information including exception tables, garbage collection maps, dependence information on other classes, and annotations for adaption to a new execution context,

l) computing a digest of the contents of the QSI using a predetermined secure hashing function,

m) encrypting said digest of the contents of the QSI to obtain a digital signature for said digest of the contents of the QSI,

n) recording said digital signature at a predefined place in the QSI, said digital signature enabling the virtual machine to detect tampering of the QSI, and

o) repeating steps (a) through (n) for each of a specified set of classes.

Claim 38 (New): A method according to claim 37, wherein:

the step of using a QSI recorder includes the further step of using a mapping to determine a location in a directory in which to place the QSI, wherein said location includes a repository containing a class for the QSI, and a directory structure implied by a fully qualified name of the class; and

for each class loader, a fixed mapping is defined from the name of the repository holding the class to the repository holding the QSI file.